UNSTRUCTURED ARCHITECTURES CONSIDERED HARMFUL

Kevin B. Bush
Code 72000, S&T Forecasting, Assessment, and Transition
Space and Naval Warfare (SPAWAR) Systems Center – Atlantic

ABSTRACT

The use of architectures to document and explore requirements during the systems engineering process is a valuable technique. Mature enterprise and system architectures can facilitate requirement discovery, analysis, and traceability in an effective and scalable manner. This role for architectures is important. However, an overemphasis of the requirements-architecture relationship risks an inadvertent devaluation of high-level design quality. Focusing on requirement capture may cause an architect to miss opportunities to make strategic engineering trade-off decisions, thereby creating an absence of design structure. This paper discusses design structure and its relationship with system complexity and warns of the pitfalls presented by unstructured architectures.

INTRODUCTION

In a 1968 letter to Communications of the ACM, Edsgar W. Dijkstra discussed concern over an unstructured programming technique he "considered harmful" to the field of computer science [Dijkstra 68]. To start a spirited conversation, he illustrated flaws of an already widely accepted standard practice. He was famously successful: The structured programming debate that followed drove improvements in language development and programming best practices that have since spanned decades [O'Reilly 04].

In this paper we borrow from Dijkstra's approach and show how unwanted complexity can flourish in unstructured architectures in order to start a conversation on the merits of structured design in architecture development.

We assert a dual-role for architectures in the systems engineering process:

1. **Requirements development** –Architectures should be used to discover undocumented capability requirements as well as to provide operational traceability for functional requirements.
2. **System-wide structural design –** Architectures should embody system-wide design decisions (*structures*) that best organize component parts to enable achievement of high-level quality attributes.

An architecture can provide traceability for a decomposed requirements list and has an integral relationship with the requirements development process. As captured in the first role, the practice of architecture development can and should ensure that system solutions deliver desired capabilities. Clearly this role for architectures is important and should not be overlooked. However, we believe an architect's attention must be equally focused on the quality of system design. As captured in the second role, an architecture should embody system-wide design decisions that leverage component organizational structures that best position the system to meet high-level quality (system-scope) attributes.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
03 OCT 2012	**Technical**	**00-00-2012 to 00-00-2012**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Unstructured Architectures Considered Harmful	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
Kevin Bush	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Space and Naval Warfare Systems Center, Atlantic,PO Box 190022,North Charleston,SC,29419	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The use of architectures to document and explore requirements during the systems engineering process is a valuable technique. Mature enterprise and system architectures can facilitate requirement discovery, analysis, and traceability in an effective and scalable manner. This role for architectures is important. However, an overemphasis of the requirements-architecture relationship risks an inadvertent devaluation of high-level design quality. Focusing on requirement capture may cause an architect to miss opportunities to make strategic engineering trade-off decisions, thereby creating an absence of design structure. This paper discusses design structure and its relationship with system complexity and warns of the pitfalls presented by unstructured architectures.

15. SUBJECT TERMS

Systems Architecture; Enterprise Architecture; Systems Engineering; Complexity

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**10**	**6**	

One can discriminate qualitatively between two satisfactory architectures – that is, architectures which satisfy capability requirements – by analyzing the degree to which they meet the quality attributes prescribed by the stakeholders. This is an important distinction. In fact, one authority considers a good architecture to be a high-level design that facilitates a system to meet its functional, quality attribute, and life cycle requirements [Clements 02]. We believe that there can be a design tension between system capabilities and system structure, and argue a balanced approach for architecture practitioners that emphasizes good "engineering taste" when faced with decisions within this trade-space.

BACKGROUND

We assume a familiarity with systems engineering, the "standard processes" approach to engineering, and the Depart of Defense Architecture Framework (DODAF) [DODCIO 2010].

We discriminate between three kinds of architectures:

- **Enterprise Architecture** – A model of a business as a system. It connects the key requirements of a business with the organizational components that perform those duties, and describes the structure of the organization with respect to its business process execution.
- **Platform/System of System Architecture** – A model of a collection of systems as a system. The structure of task-oriented systems synthesized together with pooled resources in order to create a richer spectrum of functionality and performance.
- **System/Solution Architecture** – A model of any engineered product as a system. The conceptual, operational, or physical models that describes the structure and behavior of an engineered system.

It is worthy of note that, unlike much of the engineering community, we do not discriminate between system and software architectures. While the unique characteristics of any discipline (among many, software is just one) will be reflected in unique engineering process lifecycle idiosyncrasies, for the purposes of the following discussion we make a simplifying assumption and treat them all as generic system architectures in order to facilitate a more universal discussion.

DISCUSSION

An enterprise architecture can complement traditional requirements throughout the systems engineering process and should be tightly integrated with all requirements development, analysis, and assessment activities. By documenting user activities, mapping them to system functions, and assigning functions to components, system architectures provide a rigorous means by which to develop and maintain traceability from user-prescribed capabilities all the way to testable system components. Additionally, architects are uniquely positioned to study the interactions among requirements and conduct requirements analysis in aggregate. In that way, architecture development should help to advance requirements development by exposing hidden, implied, and otherwise unarticulated user needs and by providing more precise feedback to the customer.

Because of this tremendous ability to complement requirements development and therefore improve overall system alignment with capability requirements, many architects focus their efforts to participate in requirements development and connect their products with user requirements. We believe, however, that equal attention must be given to the touch points between system architecture descriptions and downstream component designs. Too much focus on requirements can introduce unintended complexity to the overall system design resulting in timeline delays, increased downstream levels of effort, and a degradation of high-level quality attributes (such as

supportability, modifiability, and testability). Arguably, the savvy architect, not the program officer, is best suited to weigh the tradeoffs in requirement adoption as they affect overall system performance, and is best prepared to present material justification to resist any feature creep that threatens to undermine the fundamental *design structures* already applied.

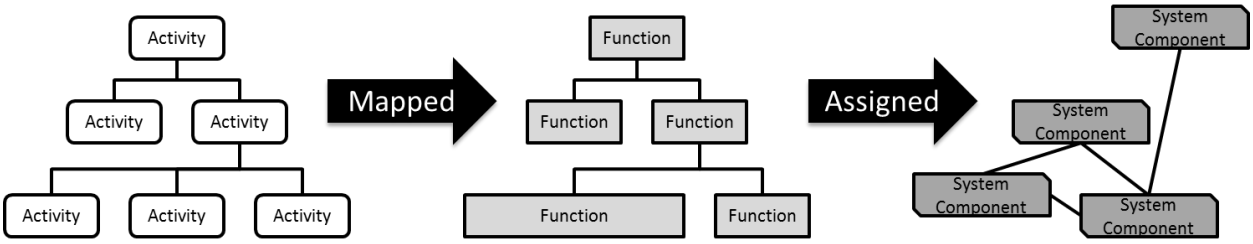

FIGURE 1 - ILLUSTRATION OF A TRADITIONAL ROLE FOR ARCHITECTURES, PROVIDING TRACEABILITY FROM USER ACTIVITIES TO FUNCTIONAL REQUIREMENTS AND THE SYSTEM COMPONENTS WHICH SATISY THEM. NOTE THE VARIATION IN STRUCTURE AT THE VARYING LEVELS OF ABSTRACTION. A LOWER-LEVEL STRUCTURE MAY BECOME SUBOPTIMIZED WHEN HIGHER-LEVEL REQUIREMENTS ARE CHANGED (E.G. FEATURE CREEP).

DESIGN STRUCTURE

A *design structure* is any system-wide pattern or regularity in system arrangement. Organizationally speaking, design structures drive homogeneity into system components and create regularities among component relationships. Dynamically speaking, design structures create temporal correlations among an architecture's components and shape system behavior patterns that facilitate tight orchestration of component functionality. Put another way, *design structure* is the fundamental mechanism by which an architect non-arbitrarily connects otherwise independent parts into a greater whole.

For any given user-prescribed capability set, one can imagine a large solution space where each individual system-solution is uniquely identified by the architecture that describes it. In that way, every architecture (solution) is a design which satisfies a particular capability set (problem), and a valid architecture is one that satisfies the unique capability requirements of a given customer. Unfortunately, architects too often focus on this first-order concern of validity and fail to qualitatively discriminate between valid architectures.

To qualitatively evaluate architectures, additional criteria must be introduced to the engineering process that can facilitate comparative analysis between valid architectures. We call these criteria high-level quality attributes [Clements, '02]. High-level quality attributes are any system-wide property that is solution agnostic. Examples include modifiability, testability, and usability. We recognize that *design structures* have characteristic qualities across the spectrum of high-level quality attributes and the two are invariably linked. Assuming many architectures will meet any given capability requirements, selection of one valid architecture over another can be based on how well they individually achieve quality attribute goals. It is in this way that an architect's scope of practice must be expanded: Architects must evaluate architectures with respect to their high-level quality attributes, and therefore must focus on the *design structures* that enable achievement of those attributes.

COMPLEXITY

The relatively new field of complex systems science provides a novel approach to the problems faced by enterprise and system architects. Complex system models provide, among other things, a new paradigm in which to view architecture development and provide objective criteria for architecture evaluation. First, a definition:

A system, ***s***, comprising a set of *Parts, each individually* ***p****, w*here **Ω** represents the number of states expressible by the component, is complex *if and only if*:

$$\boldsymbol{\Omega}(p) < \boldsymbol{\Omega}(s) < \prod_{p \in Parts} \boldsymbol{\Omega}(p)$$

That is to say that a system synthesized out of a set of parts is complex if the number of possible states of the system is more than the number of states of any of its parts, but less than the number of states of the aggregated parts. Intuitively, if a system is no more complex than any of its parts, then it is exhibiting perfectly coherent behavior (e.g. a crystalline lattice); if a system is no less complex than the product of its parts, then it is exhibiting perfectly independent behavior (e.g. an ideal gas) [Bar-Yam '03].

> **Note:** Virtually all engineered systems today exhibit some degree of design *structure* that orchestrates the functions of otherwise *independent* parts, thereby satisfying this condition. Put another way: All engineered systems are necessarily complex, and the approaches of complex system science are of particular relevance.

In follows that the degree of complexity a system exhibits is inversely proportional to the degree of structure it maintains. And with a satisfactory objective measure of complexity, savvy architects can qualitatively evaluate design decisions by analyzing impact to overall system complexity for each option, therefore wielding *design structures* as a tool to reduce complexity where and when necessary. A careful balance must be made, however, as too much uniformity can create systemic risks inherent to severe homogeneity.

Because *design structure* and complexity are inherently related, and *design structures* each have inherent characteristic high-level quality attributes, we can use objective complexity measures to also identify relative degrees of quality in enterprise and system architectures. Good architectures will exhibit strong *design structures* and therefore **Ω(*s*)** will approach **Ω(*p*)**. In contrast, for systems designed without consideration to quality attributes and *design structure*, **Ω(*s*)** and **Ω(*p*)** will diverge, complexity will flourish as coherence of the system is lost, and programmatic priorities such as budget, time, and performance will be compromised. This issue of quality is even more poignant when considered beyond the scope of a single system, but a product line of systems that share common core structures.

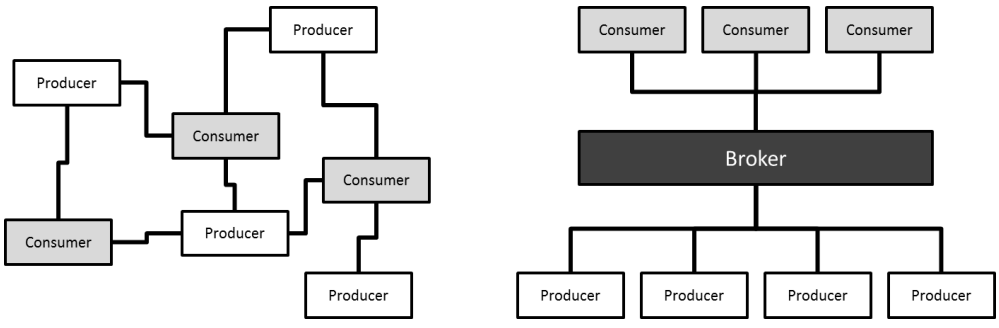

FIGURE 2 - ILLUSTRATION OF TWO ARCHITECTURES THAT VALIDATE THE SAME FUNCTIONAL REQUIREMENTS. THE SYSTEM ON THE LEFT HAS MANY PAIR-WISE INTERFACES THAT DOWNSTREAM ENGINEERS MUST DESIGN AND MAINTAIN. USING THE SIMPLE BROKER-MODEL, THE SYSTEM ON RIGHT EXHIBITS A STRONG STRUCTURE AND HAS MINIMIZED INTERFACE COMPLEXITY.

ADDITIONAL THOUGHTS

Complex systems science also provides a useful framework through which to better understand the role of self-organization and emergent system behavior. Descartes first hypothesized that system dynamics can lend order (structure) to a system in his 1637 treatise "*Discourse on the Method*", and Kant first used the term 'self-organizing' in his "*Critique of the Power of Judgment*" in 1790. But scholars have only recently begun to argue that developing systems from the bottom-up may better meet capability requirements than the traditional top-down approach [Forester, '61]. We characterize this as the "orchestrated design vs self-organization problem" and recognize both that (1) self-organization is intrinsically a more scalable approach to engineered systems, and that (2) early successes in fields such as synthetic biology have already been made in the area of self-organizing systems engineering. One needs to look no further than any sufficiently large organizational process to identify an active (and most likely accidental) experiment in self-organization. Enterprise Architectures are often either unenforced or too ambiguous to convey any meaningful top-down structure – rather, learning organizations adapt to changes in their environments in an ad-hoc and bottom-up manner. It is our expectation that future developments in the area of self-organizing systems will lend themselves to more practical and cross-disciplinary engineering applications.

We note that one can interpret *design structure* in several ways. This paper has considered system structure in an information-theoretic frame, couching it as either *coherence among parts*, or *correlation among system component states*. An alternative interpretation comes from control theory – the law of requisite variety states that a control system can be stable only if the number of states of its control mechanism exceeds the number of states of the system being controlled [Ashby '56]. From a systems architecture perspective, that means that a stable system must at least be prepared to handle all of the environmental conditions it must face. This insight bounds the degree of structure that we can design into a system (i.e. too much structure reduces the systems state space below that of its environment and makes automation infeasible). While this paper has argued for an increased emphasis on *design structure*, we acknowledge this fundamental limit and encourage system architects to exercise good design taste when scoping system environments and balancing *structure* with *functionality*.

Structure can also be interpreted as predictability. While many engineers intuitively understand how deterministic behavior can arise out of stochastic processes (e.g. via a mean field assumption), too many falsely believe that stochastic random behavior cannot arise out of deterministic processes. One needs to look no further than the logistic map recurrence relation to observe the emergence of chaotic behavior out of a deterministic and discrete mathematical function [May, '76]. This suggests an increased emphasis on the role of scale selection in the architecture development process, and hints that a sweet spot for design structure can always be found, even nestled between lower and upper scales with apparently chaotic behavior characteristics.

Finally, we believe future enterprise and system architects within the defense community may benefit from the following observation: Component orchestration and functional emergence are symmetric equivalents that traverse the levels of warfare. Orchestration drives structure from the strategic to the tactical level, and emergence percolates structure from the tactical to the strategic. Since emergence happens with or without intent, we must be mindful of the emergent consequences of our design decisions.

REFERENCES

1. Bar-Yam, “Unifying Principles in Complex Systems in Converging Technology (NBIC) for Improving Human Performance”, M. C. Roco and W. S. Bainbridge eds., Kluwer, 2003
2. Ashby, “An Introduction to Cybernetics”, Chapman & Hall, 1956
3. Clements & Kazman & Klein, “Evaluating Software Architectures”, Westford Massachusetts: Addison-Wesley, 2002
4. DODCIO, “DoD Architecture Framework Version 2.02”, http://dodcio.defense.gov/dodaf20.aspx, August 2010
5. Dijkstra, “Go To Statement Considered Harmful”, Communications of the ACM, Vol. 11, No. 3, pp. 147-148, March 1968
6. Forester & Pask & Beer & Weiner, “Cybernetics: or Control and Communication in the Animal and the Machine”, MIT Press, 1961
7. Lenahan & Pacetti & Heller & Reed & Mori, “Interoperability Risk Mitigation through the Application of Operational Capability Based Engineering”, 14th International C2 Research and Technology Symposium, 2009
8. May, “Simple mathematical models with very complicated dynamics”, Nature 261, 1976
9. O’Reilly, “History of Programming Languages”, O’Reilly Media Inc., 2004